# QLC-3) Guided Solution

1. We'll draft out a test defining what we think we want the production code to do

```
2. public class TopicScoreWriterTest
  {
     @Test
     public   void  verify_topic_score_details_written_out_once()
     {
        // arrange
        String physics = "Physics";

        ArrayList<TopicTopScore> topTopScores = new ArrayList<>();
        topTopScores.add( new TopicTopScore(physics, 89));

        TopicScoreWriter cut = new TopicScoreWriter();

        // act
        cut.writeScores( topTopScores );

        // assert
     }
  }
```

3. Line 12 represents the implementation class. We will call a method called `writeScores()` passing an array `TopicTopScores`. Since this method will not return a result that is testable in this context, how are we going to test the result of the `writeScrores()`, it's going to make a call to a Java API. We are not going to test the Java API call, many millions of developers have already done this. We can use the mocking libraries `verify` capability. But what are we going to verify?
4. We'll begin by decoupling the Java API call to write to the file stream from the TopicScorWriter. To do this we could pass into the constructor a `java.io.FileWriter` object or something similar. But this would tie the production too tightly to java.io.FileWriter. Instead, let's pass an object that encapsulates the file writer functionality. This object needs to support two capabilities 1) open a text file to write to, and 2) write the formatted string to the opened text file. With this design in mind we can update our test and then dig a little deeper into the implementation code.
5. Our updated test class now looks like this

6. 
```
public class TopicScoreWriterTest
  {
     interface IFileWriter
     {
        void  writeLine( String lineToWrite );
     }

     @Test
     public   void  verify_topic_score_details_written_out_once()
     {
        // arrange
        String physics = "Physics";
        String art = "Art";
        String compSci = "Comp Sci";
        String expectedResult = "Physics, 89";

        ArrayList<TopicTopScore> topTopScores = new ArrayList<>();
        topTopScores.add( new TopicTopScore(physics, 89));

        String fileToWrite = "testfile.txt";
        IFileWriter fileWriter = mock(IFileWriter.class);

        TopicScoreWriter cut = new TopicScoreWriter( fileWriter );

        // act
        cut.writeScores( topTopScores );

        // assert
        verify( fileWriter, times(1)).writeLine(expectedResult);
     }
  }
```

7. Notice the use of the interface at line 3. We then use this interface to create Mock object at line 21. At line 29 wr are going to verify that the method writeLine() is called once with a string formatted as shown at line 15.
8. Move the IFileWriter interfaces into its own file

9. 
```
package com.s2s.demos.topicmanager;

  public interface IFileWriter
  {
        void  writeLine( String lineToWrite );
  }
```

10. Let's write the implementation code for `TopicScoreWriter`. We've used a canned result (line 20) to test our theory

11. 
```
package com.s2s.demos.topicmanager;

import java.util.ArrayList;

/**
 *
 * @author selvy
 */
public class TopicScoreWriter
{
   private IFileWriter itsFileWriter;

   public TopicScoreWriter(IFileWriter fileWriter )
   {
      this.itsFileWriter = fileWriter;
   }

   void writeScores(ArrayList<TopicTopScore> topTopScores)
   {
      itsFileWriter.writeLine("Physics, 89");
   }
}
```

12. Run the test and it should pass
13. We'll now update `writeScores()` so that it writes the data being passed

14. 
```
   void writeScores(ArrayList<TopicTopScore> topTopScores)
   {
      if( topTopScores.size() > 0 )
      {
         TopicTopScore tts = topTopScores.get(0);
         String dataToWrite = tts.getTopicName() + ", " + tts.
getTopScore();
         itsFileWriter.writeLine(dataToWrite);
      }
   }
```

15. The if statement introduces a new logic path in the code, we should have test for this

16.
```
@Test
public   void  verify_topic_score_details_not_written()
{
   // arrange

   ArrayList<TopicTopScore> topTopScores = new ArrayList<>();

   IFileWriter fileWriter = mock(IFileWriter.class);

   TopicScoreWriter cut = new TopicScoreWriter( fileWriter );

   // act
   cut.writeScores( topTopScores );

   // assert
   verify( fileWriter, times(0)).writeLine(Mockito.any());
}
```

17. Finally, add another test that writes more top scores for different topics

18.
```
    @Test
    public   void
  verify_topic_score_details_written_out_multiple_times()
    {
       // arrange
       String physics = "Physics";
       String art = "Art";
       String compSci = "Comp Sci";
       String physicsResult = "Physics, 89";
       String artsResult = "Art, 87";
       String comSciResult = "Comp Sci, 97";

       ArrayList<TopicTopScore> topTopScores = new ArrayList<>();
       topTopScores.add( new TopicTopScore(physics, 89));
       topTopScores.add( new TopicTopScore(art, 87));
       topTopScores.add( new TopicTopScore(compSci, 97));

       String fileToWrite = "testfile.txt";
       IFileWriter fileWriter = mock(IFileWriter.class);

       TopicScoreWriter cut = new TopicScoreWriter( fileWriter );

       // act
       cut.writeScores( topTopScores );

       // assert
       verify( fileWriter, times(1)).writeLine(physicsResult);
       verify( fileWriter, times(1)).writeLine(artsResult);
       verify( fileWriter, times(1)).writeLine(comSciResult);
    }
```

19. And the implementation code has been updated to

```
20. public class TopicScoreWriter
  {
     private IFileWriter itsFileWriter;

     public TopicScoreWriter(IFileWriter fileWriter )
     {
        this.itsFileWriter = fileWriter;
     }

     void writeScores(ArrayList<TopicTopScore> topTopScores)
     {
        for(TopicTopScore tts : topTopScores)
        {
           String dataToWrite = tts.getTopicName() + ", " + tts.
  getTopScore();
           itsFileWriter.writeLine(dataToWrite);
        }
     }
  }
```